Algorithms, Architectures and Circuits for Low Power HEVC Codecs

by

Chiraag Juvekar

B.Tech., Indian Institute of Technology Bombay (2012)
M.Tech., Indian Institute of Technology Bombay (2012)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

© Massachusetts Institute of Technology 2014. All rights reserved.

Signature redacted

Author ........................................

Department of Electrical Engineering and Computer Science

May 21, 2014

Signature redacted

Certified by ..............................

Anantha P. Chandrakasan

Joseph F. and Nancy P. Keithley Professor of Electrical Engineering

Thesis Supervisor

Signature redacted

Accepted by ..............................

Leslie A. Kodolziejski

Chairman, Department Committee on Graduate Theses
Algorithms, Architectures and Circuits for Low Power

HEVC Codecs

by

Chiraag Juvekar

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

In order to satisfy the demand for high quality video streaming, aggressive compression is necessary. High Efficiency Video Coding (HEVC) is a new standard that has been designed with the goal of satisfying this need in the coming decade. For a given quality, HEVC offers 2x better compression than existing standards. However, this compression comes at the cost of a commensurate increase in complexity.

Our work aims to control this complexity in the context of real-time hardware video codecs. Our work focused on two specific areas: Motion Compensation Bandwidth and Intra Estimation. HEVC uses larger filters for motion compensation leading to a significant increase in decoder bandwidth. We present a novel motion compensation cache that reduces external memory bandwidth by 67% and power by 40%. The use of large, variable-sized coding units and new prediction modes results in a dramatic increase in the search space of a video encoder. We present novel intra estimation algorithms that substantially reduce encoder complexity with a modest 6% increase in BD-rate. These algorithms are co-designed with the hardware architecture allowing us to implement them within reasonable hardware constraints.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Acknowledgments

This thesis would not have been possible without help and advice of many individuals. Firstly, I would like to thank Prof. Anantha Chandrakasan, my research advisor for his mentorship and for providing me an opportunity to work on this challenging and interesting problem.

I would like to thank Mehul Tikekar and Chao-Tsung Huang, my colleagues on this project. They were the ones who first introduced me to the wonderful and complex world of video coding and I have learned much from them. It has been great fun working with you guys.

Prof. Vivienne Sze has been an invaluable resource in working on this project. Her deep understanding of the nitty-gritties of video coding has helped scope out solutions to various unexpected issues that have come up.

Discussions with various members of ananthagroup including Arun, Nachiket and Priyanka have helped me better understand a wide variety of circuits and algorithms related concepts. Thanks for patiently answering all my silly questions.

I would like to thank TSMC University shuttle program for chip fabrication support. I would also like to thanks Texas Instruments and the Stata Family for funding this research.

Finally, I would like to thank Sai and my parents, Rashmi and Shashikant. If not for their constant support and motivation, I would not have made it thus far.
## Contents

1 Introduction ............................................. 15
   1.1 Background and Motivation .............................. 15
   1.2 HEVC Overview and Challenges ............................ 17
      1.2.1 Prediction in HEVC .................................. 19
   1.3 Thesis Contributions .................................... 24
      1.3.1 Motion Compensation Cache .......................... 24
      1.3.2 Hardware Friendly Intra Estimation .................. 24

2 Motion Compensation Cache ............................... 27
   2.1 Memory System Overview ................................ 27
      2.1.1 Target DRAM System ................................ 30
   2.2 Memory Address Mapping ................................ 30
      2.2.1 Cache line geometry for minimizing fetch of unused pixels 32
      2.2.2 Twisted mapping for minimizing row precharge and activation 33
      2.2.3 Minimizing conflict misses .......................... 34
   2.3 Motion Compensation Cache Design ....................... 34
      2.3.1 Four-Parallel Data Flow ....................... 36
2.3.2 Queue Management and Hazard Control ........................................ 36
2.4 Results .................................................................................................. 38
  2.4.1 Cache Specification ......................................................................... 38
  2.4.2 DRAM Bandwidth and Power Savings ........................................... 38
2.5 4k UltraHD Demo Setup ....................................................................... 40

3 Hardware Friendly Fast Intra Estimation ............................................. 45
  3.1 Current Approaches ........................................................................... 46
    3.1.1 HM Reference Implementation ..................................................... 46
    3.1.2 Related Work ............................................................................. 49
  3.2 Algorithm Design ................................................................................ 50
    3.2.1 Gradient Based Mode Pruning ..................................................... 51
    3.2.2 Fast Mode Decision .................................................................... 54
    3.2.3 Results ....................................................................................... 56
  3.3 Architecture and Implementation ....................................................... 58
    3.3.1 Challenges and Design Goals ..................................................... 60
    3.3.2 Tiles, Multi-threading and Feedback ......................................... 62
    3.3.3 Top Level Pipeline ..................................................................... 63
    3.3.4 Prefetch and Original Pixel SRAM ............................................. 66
    3.3.5 Gradient based Mode Pruning Architecture and Circuits ............ 68
    3.3.6 Mode Decision Architecture ..................................................... 74

4 Conclusion ............................................................................................... 77
  4.1 Thesis Summary .................................................................................. 77
  4.2 Future Work ....................................................................................... 79
List of Figures

1-1 Successive 2×increases in compression and complexity with new coding standards ........................................... 16
1-2 Major components of a generic HEVC encoder and decoder ................................................................. 17
1-3 An example CTU split that can be encoded by an HEVC encoder ...................................................... 21
   (a) CTU Split ................................................................. 21
   (b) Quad Tree ................................................................. 21
1-4 Illustration of fractional motion compensation and possible angular intra modes in HEVC .................... 23
   (a) Fractional Motion Compensation .................................... 23
   (b) Intra modes ................................................................. 23
2-1 Example MC cache dispatch order ................................................. 28
2-2 Latency Aware DRAM mapping ....................................................... 31
2-3 Four-parallel cache architecture .................................................... 35
2-4 Cache hit rate as a function of cache paramters ..................................................... 37
   (a) Cache line Geometry ................................................. 37
   (b) Cache Size ................................................................. 37
(c) Cache Associativity ........................................... 37

2-5 Comparison of DDR3 bandwidth and power consumption ........ 39
   (a) Bandwidth Comparison ...................................... 39
   (b) Power Comparison ........................................... 39
   (c) BW across sequences ....................................... 39

2-6 Block diagram of the 4k UltraHD real-time HEVC decoding demo ... 41

2-7 The die micrograph showing the MC Cache and the complete demo
     systems showing the decoder chip attached to the ML605 board ... 43
   (a) Die Micrograph ............................................. 43
   (b) Demo System ................................................ 43

3-1 An illustration of the gradient computation algorithm ............ 52

3-2 Distribution of intra modes estimated by HM. Modes included by
     default in fast mode decision are shown in red .................. 54

3-3 PSNR vs. Rate comparison of our algorithm (green) with HM (blue) 57

3-4 Example video sequence with 4 tiles .......................... 62

3-5 Effect of Multi-threading on throughput. The top figure shows a single
     threaded implementation and demonstrates the stall caused by the
     neighbor dependency. Using 4 tiles we can potentially achieve 100%
     utilization by processing a different tile in the stall period as seen in
     the middle figure. The real utilization is lower because the variable
     PU size prevents perfect pipelining as shown in the bottom figure ... 64

3-6 Top level of the intra estimation block showing the various computa-
     tion blocks and memories .................................... 65
LIST OF FIGURES

3-7 Pipeline diagram for the top level showing the original pixel prefetch, gradient based mode pruning and mode-decision blocks 66
3-8 An example of a misaligned 6 pixel gradient based mode pruning access 67
3-9 The multiple read pattern required to support both RMD and RDO operations need 8 SRAM banks with 4 pixel readouts. Using 2 banks for RMD and a separate buffer for RDO offers a better area trade-off between cell area and read/write circuits in the SRAM 68
3-10 Gradient based mode pruning design consisting of histogram and reuse registers, a separate original pixel SRAM and computation blocks 69
3-11 Deployment of vertical and horizontal original pixel reuse registers 70
3-12 Effective SRAM read overhead for a 16×16 PU 71
3-13 Parallel computation of 4 vertical and horizontal gradients 72
3-14 Multiple constant multiplication based mode computation 73
3-15 Selection of the 2 best modes over 2 cycles 74
3-16 Pipeline schedule for gradient based mode pruning (the 16×16 and 32×32 PU computations are scheduled only after their last 8x8 PUs) 74
3-17 Mode decisions architecture showing the RMD and RDO blocks. The DCT related blocks are marked separately 75
LIST OF FIGURES
List of Tables

1.1 Comparison of the coding tools in HEVC and AVC/H.264 20

2.1 Fractional Motion Compensation Overheads for an 8x4 pixel DRAM minimum access unit 29

2.2 Average fetch overhead as function of cache line geometry 32

2.3 Comparison of Twisted 2D Mapping and Direct 2D Mapping 33

2.4 Overview of MC Cache Specifications 38

3.1 Comparison of published literature on intra estimation algorithms 49

3.2 Comparison of BD rate across various QPs and sequences 59

3.3 BD rate comparison of our algorithm with [1] and [2] 60

3.4 Comparison of OrgPel SRAM configurations 69
LIST OF TABLES
Chapter 1

Introduction

1.1 Background and Motivation

Video traffic continues to be a major driver for today’s internet. At the end of 2012, video accounted for 51% of the global mobile internet traffic and it is expected to grow to 66% by 2017. [3]

Common mobile applications for online video streaming like YouTube, Netflix, etc. download the compressed bitstream using either 3G, Wi-Fi or LTE radios. The average energy cost of a downloading a compressed bit of video is 100nJ [4] for current commercial radios. On the other hand decompressing this bit requires just 4nJ of energy[5]. This factor of 25× in the energy cost of the two operations clearly motivates the use of higher complexity algorithms that provide better compression. High Efficiency Video Coding (HEVC) was ratified as an international standard in January 2013 to address this issue. As seen in figure 1-1, HEVC follows the existing trend of offering 2× better compression over the previous generation of video codecs.
while retaining the same visual quality [6]. This coding gain comes at the cost of a commensurate 2× increase in the decoder complexity [7]. Innovative circuit techniques are required to handle the increased complexity of these algorithms while maintaining real-time operation and support for ever-increasing resolutions.

Figure 1-1: Successive 2× increases in compression and complexity with new coding standards

In contrast, building real-time encoders is an entirely different ball game. Achieving the 2× compression promised by HEVC is about 10-100× as complex as decoding the same compressed bitstream. [7] If algorithm complexity is not controlled then encoding the video might become as expensive as transmitting it. This is particularly important for applications like video conferencing where the encoded video will be transmitted exactly once and thus we cannot amortize the encoding cost over multiple transmissions. For these applications, innovative algorithms and architectures are required to simplify the encoder without incurring excessive coding penalty.

Our work addresses both these issues in the context of two specific problems
for HEVC: motion compensation bandwidth and intra estimation complexity. In the following sections, we provide an overview of HEVC and explain these issues in detail.

1.2 HEVC Overview and Challenges

Like AVC/H.264, HEVC too is a block-based video coding standard. This means that every video frame is tiled by square blocks called Coding Tree Units (CTUs) which are encoded in a raster scan fashion. HEVC allows three CTUs sizes: 64×64, 32×32 and 16×16. Each CTU is recursively split into smaller blocks called coding units (CU). The largest CU can be as large as the CTU itself whereas the smallest CU can be up to 8×8 pixels in size. As shown in figure 1-2, a basic scheme of estimation, prediction (intra/inter), transform, loop filtering and entropy coding is employed to encode these CUs.

Figure 1-2: Major components of a generic HEVC encoder and decoder

The prediction process generates an estimate of the video based on previously
decoded pixels. HEVC offers compression by transmitting the method used for prediction (prediction mode) rather than the actual raw pixels in the video. These predictions are performed on the granularity of prediction units (PUs). A PU can be the same size as a CU or a CU may be split into two or four smaller PUs. The HEVC prediction process and the available prediction modes are explored in detail in section 1.2.1.

When compressing a video, the estimation block in the encoder figures out which PU sizes and predictions modes will offer maximum compression. Prediction and estimation are complementary processes.

Since the prediction process is not perfect, the predicted pixel values in a PU differ from the values of the original pixels. In order to preserve quality, the encoder also transmits the prediction error to the decoder.

The human eye is not very sensitive to high frequency components of the prediction error when viewing the video sequence at a reasonable frame rate. Additional compression is achieved by discarding these components. This is achieved by transforming the prediction error and quantizing it before transmission. The transform concentrates the energy in low frequency components. Further compression is achieved by using run-length coding on these transformed values.

Both the coding mode and the quantized coefficients are entropy coded using a lossless context-adaptive binary arithmetic coder (CABAC) before being written out to the final bitstream. The decoder reverses these steps sequentially to undo the entropy coding, quantization and transform to recover the prediction error (residue). It then adds it back to the prediction that it generates based on the mode information and is able to reconstruct the original video sequence.
1.2. **HEVC OVERVIEW AND CHALLENGES**

This reconstructed video is not perfect since loss is introduced due to the quantization process. Thus, video coding process exhibits an inherent rate-distortion (RD) trade-off. Choosing a higher quality quantization may use more bits in the bitstream (higher rate) but it is possibly to achieve low distortion. Conversely using coarser quantization one can reduce the bitstream size at the cost of video fidelity. This trade-off is controlled by a quantization parameter (QP) with larger values of QP corresponding to coarser quantization.

The encoding problem is to choose coding decisions (CU splits, prediction modes, etc.) that are RD optimal. The encoder is allowed reduce complexity by choosing to not implement certain modes that it believes are not important to getting the best RD trade-off. The decoder on the other hand does not have this flexibility. Since a decoder has to be compatible with any encoded bitstream, it must support all modes including those that may result in a large complexity overhead but not provide commensurate coding gain.

Each of these individual steps is more complex when compared to their AVC counterparts [8]. Table 1.1 compares the two standards. This work focusses on the intra prediction and inter prediction blocks, which are explained in the following section.

### 1.2.1 Prediction in HEVC

HEVC allows for a complex hierarchical split structure due to the quad-tree partitioning. An example of a 64×64 CTU split structure is shown in 1.2.1. The quad-tree for the bottom-left 32×32 CU is shown in 1-3(b).

Every leaf node of the tree is a PU and can be predicted in one of two ways.
CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>Tool</th>
<th>HEVC</th>
<th>AVC/H.264</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Block Sizes</td>
<td>CTU: 64×64, 32×32 and 16×16</td>
<td>MB: 16×16</td>
</tr>
<tr>
<td>Coding Unit (CU)</td>
<td>4 sizes: 64×64 to 8×8</td>
<td>3 sizes: 16×16 to 4×4</td>
</tr>
<tr>
<td>Prediction Units (PU)</td>
<td>24 types (per CU)</td>
<td>7 types (per MB)</td>
</tr>
<tr>
<td>Transform Units (TU)</td>
<td>DCT: 32×32 to 4×4, DST: 4×4</td>
<td>8×8 and 4×4</td>
</tr>
<tr>
<td>Intra Prediction</td>
<td>35 modes</td>
<td>10 modes</td>
</tr>
<tr>
<td>Inter Prediction</td>
<td>8-tap filters</td>
<td>6-tap filters</td>
</tr>
<tr>
<td>Loop Filter</td>
<td>Deblocking, SAO</td>
<td>Deblocking</td>
</tr>
<tr>
<td>Entropy Coding</td>
<td>CABAC</td>
<td>CABAC/CAVLC</td>
</tr>
<tr>
<td>Parallelism Modes</td>
<td>Tiles, Wavefronts</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1.1: Comparison of the coding tools in HEVC and AVC/H.264

- **Inter Coding:**

  Video sequences have a lot of temporal redundancy. Pixels in a given frame are very similar to those in the preceding and succeeding frames. Inter prediction exploits this and predicts a given block of pixels based on those in an already decoded frame. Due to the motion of the subject or the camera, the referenced region may be slightly offset from the pixels in the current frame. This offset is called a motion vector (MV). This offset can either be an integral number of pixels (integer MV) or up to quarter-pixel accurate (fractional MV).

  The operation of computing the predicted pixels based on the referenced frames and MV values is called motion compensation. If a given PU uses an integer MV, then motion compensation merely involves copying the relevant pixels. Fractional MVs are compensated by interpolating the pixels surrounding the referenced region. HEVC uses an 8-tap filter to perform this interpolation as shown in figure 1-4(a).
1.2. HEVC OVERVIEW AND CHALLENGES

(a) CTU Split

(b) Quad Tree

Figure 1-3: An example CTU split that can be encoded by an HEVC encoder
HEVC allows predicting a single PU from two previously decoded frames (bi-prediction) which themselves may be selected from up to 16 frames stored in the decoded picture buffer. In the low-delay (LD) mode, both these frames refer to past frames. In the random access (RA) mode, the frames may be re-ordered and the decoding order and display order may differ. Thus, a given frame can also be predicted from a future frame. Inter coding in HEVC supports multiple non-square PU configurations. Using these features, the predicted pixel values can be made closer to the original pixels. This results in lower prediction error and thus leads to better compression.

- **Intra Coding:**
  Intra coding exploits spatial redundancy within a given frame. Pixels in a given frame are predicted based on previously encoded neighboring pixels in the same frame. To provide improved coding gain for high resolutions, HEVC provides 35 intra modes compared to the 9 modes available in AVC/H.264. [9]. As shown in figure 1-4(b), 33 of the total 35 modes correspond to angular intra modes. For these modes, the neighboring pixels are interpolated along the specified direction to predict the current PU. In addition, HEVC also provides DC and Planar mode (labelled 0 and 1 respectively) analogous to AVC/H.264. The angular modes are labelled 2-34 in the clockwise direction with the horizontal and vertical modes corresponding to 10 and 26 respectively. A luma block may signal of any of the 35 intra modes. Chroma blocks are restricted to one of 5 modes: 0, 1, 10, 26 and either the luma mode corresponding to the same CU or mode 34.
AVC/H.264 only allows for left, top and top-right neighbors. This is because in the 16×16 macroblock coding structure of AVC/H.264 the bottom-left neighbors are rarely available. HEVC allows the use of these neighbors since the quad-tree scheme ensures bottom-left pixels are available in a larger fraction of PUs. HEVC also allows a mode dependent filter operation to be performed on the neighbors prior to their use for interpolation.
1.3 Thesis Contributions

1.3.1 Motion Compensation Cache

HEVC uses 8-tap interpolation filters compared to the 6-tap filters in AVC/H.264. Thus, an N×N block of pixels in the current frame may need up to \((N+7) \times (N+7)\) pixels for fractional interpolation. Bi-prediction serves to double this requirement. For example bi-predicted 8×8 block in the current frame needs 450 reference pixels. This is already a 7× increase. Memory related quantization could add to this overhead significantly as seen in section 2.1.

It is infeasible to provide such a high throughput memory interface. Moreover, the power consumed by such an interface would be prohibitive for mobile applications. We propose a motion compensation cache to control this bandwidth and allow for real-time operation with a reasonable power budget. We integrate this cache into a HEVC decoder and show a real-time demonstration of 4k UltraHD decoding.

1.3.2 Hardware Friendly Intra Estimation

Intra prediction can occur on block sizes ranging from 4×4 to 32×32. Given a 64×64 pixel, CTU an HEVC encoder must decide both the correct hierarchical split that captures this region and one out of the 35 intra modes for each of the PU leafs in the quad-tree. This represents a significant complexity increase over existing standards like AVC/H.264 and has been acknowledged as a challenging problem for real-time encoders [7]. The reference software implementation achieves an average 22.3% lower bit-rate when compared to AVC/H.264 but is around 450× slower than real-time [7].
Implementing the reference intra encoding algorithm in hardware will incur a significant area penalty. We propose a low complexity intra estimation algorithm that ensures low coding loss. We then propose a hardware architecture to implement the same for real-time operation.

Both these solutions involve a combination of circuits, algorithms and architecture techniques and are presented in detail in the next two chapters. The concluding chapter summarizes our results and proposes directions for future research.
Chapter 2

Motion Compensation Cache

Neighboring PUs often use similar MVs. This results in a significant overlap in the reference pixel data required by neighbouring inter PUs. This overlap can be exploited by a read-only motion compensation (MC) cache. In addition to reducing the bandwidth requirement, the presence of a cache also hides the variable latency of the DRAM. This chapter describes the design of an MC cache to support real-time decoding of 4k UltraHD HEVC video [5]. We describe the overall cache architecture and the various trade-offs that led us to choose this design.

2.1 Memory System Overview

A HEVC decoder may be required to store up to 16 previously decoded frames as reference candidates. For large resolutions like 4k Ultra HD a single YUV420 frame is 12M bytes in size. External DRAM is the only viable option for the storing the decoded picture buffer at these high resolutions. The minimum access unit (MAU) of

27
Figure 2-1: The example MC cache dispatch for a $23 \times 23$ reference region of a $16 \times 16$ PU. 7 cycles are required to fetch the 28 MAU at 4 MAU per cycle. Note that dispatch region need not be aligned with the four parallel cache datapaths, thus requiring a reordering. In this example, the region starts from datapath #1.

A DRAM is typically much larger than a single pixel. Thus, when fetching reference pixels, it is inevitable that some additional unused pixels will be fetched. A sample illustration for a 32 pixel minimum access unit is shown in figure 2-1. The light grey region indicates the reference pixels needed for the fractional motion compensation and the dark grey region shows the pixels that will actually fetched from the DRAM because of the MAU quantization.

Table 2.1 shows the net fetch overhead resulting from the fractional motion compensation and MAU quantization. This computation assumes that all possible motion vectors for a given PU size are equally likely and averages over them to find the net overhead. As expected smaller PU sizes face the largest overheads. The table also shows the relative likelihood of a pixel belonging to a specific PU size as seen in the ParkJoy sequence. We use this to find the average overhead seen by the
entire video sequence. We see an average $4.54 \times$ increase in the amount of pixels fetched. These numbers assume only one reference region; bi-prediction will double this number. Half of the total overheads comes from MAU quantization.

<table>
<thead>
<tr>
<th>PU type (Likelihood)</th>
<th>Total Overhead</th>
<th>MAU fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x 4 (6%)</td>
<td>12.44</td>
<td>62.55%</td>
</tr>
<tr>
<td>8 x 4 (5%)</td>
<td>7.75</td>
<td>54.13%</td>
</tr>
<tr>
<td>4 x 8 (5%)</td>
<td>8.25</td>
<td>57.98%</td>
</tr>
<tr>
<td>8 x 8 (32%)</td>
<td>5.14</td>
<td>48.72%</td>
</tr>
<tr>
<td>16 x 8 (5%)</td>
<td>3.59</td>
<td>39.76%</td>
</tr>
<tr>
<td>8 x 16 (5%)</td>
<td>3.84</td>
<td>44.44%</td>
</tr>
<tr>
<td>16 x 16 (22%)</td>
<td>2.68</td>
<td>34.64%</td>
</tr>
<tr>
<td>32 x 16 (3%)</td>
<td>2.1</td>
<td>26.47%</td>
</tr>
<tr>
<td>16 x 32 (3%)</td>
<td>2.22</td>
<td>31.03%</td>
</tr>
<tr>
<td>32 x 32 (11%)</td>
<td>1.74</td>
<td>22.48%</td>
</tr>
<tr>
<td>64 x 32 (1%)</td>
<td>1.5</td>
<td>15.97%</td>
</tr>
<tr>
<td>32 x 64 (1%)</td>
<td>1.56</td>
<td>19.35%</td>
</tr>
<tr>
<td>64 x 64 (3%)</td>
<td>1.34</td>
<td>13.04%</td>
</tr>
</tbody>
</table>

Table 2.1: Fractional Motion Compensation Overheads for an 8x4 pixel DRAM minimum access unit.

In addition to MAU quantization, the other major issue faced with DRAM is the variable latencies observed by different requests. This is an artifact of the internal structure of the DRAM [10], as accesses to different rows in the same DRAM bank face additional latency penalties due to activate and precharge operations.
2.1.1 Target DRAM System

The target DRAM system is intended to support up to HEVC level 5 decoding and the DPB can support six 3840×1260 pixel frames. The DRAM system is composed of two 64M×16-bit DDR3 DRAM modules and has a 32 byte minimum access unit (MAU). DDR3 SDRAM has 8 banks per module. Each bank addresses 4K rows and 512 MAUs can be stored within each row.

2.2 Memory Address Mapping

An ideal mapping of pixels to DRAM addresses should minimize both the number of DRAM accesses and the latency experienced by each access. These two goals can be achieved by minimizing the fetch of unused pixels and the number of row precharge/activate operations respectively. Note that this optimization affects only how the pixels are stored in DRAM and can be performed even in the absence of a MC cache. The DRAM addresses in turn must be mapped to cache lines such that conflict misses are minimized.

Figure 2-2 shows the latency-aware memory map we propose using the above guiding principles. The luma color plane of a picture is tiled by 256×128 pixel blocks in raster scan order. Each block maps to an entire row across all eight banks. These blocks are then broken into eight 64×64 blocks which map to an individual bank in each row. Within each 64×64 block, 32-byte MAUs map to 8×4 pixel blocks that are tiled in a raster scan order. In figure 2-2, the numbered square blocks correspond to 64×64 pixels and the numbers stand for the bank they belong to. Note how the mapping of 128×128 pixel blocks within each 256×128 regions alternates from left to
2.2. MEMORY ADDRESS MAPPING

Figure 2-2: Latency Aware DRAM mapping. 128 8×4 MAUs arranged in raster scan order make up one block. The twisted structure increases the horizontal distance between two rows in the same bank. Note how the MAU columns are partitioned into 4 datapaths (based on the last 2 bits of column address) of the four-parallel cache architecture.

right. Figure 2-2 shows this twisting behavior for a 128×128 pixel region composed of four 64×64 blocks that map to banks 0, 1, 2 and 3.

The chroma color plane is stored in a similar manner in different rows. The only notable difference is that an 8×4 chroma MAU is composed of pixel-level interleaving of 4×4 Cr and Cb blocks. This is done to exploit the fact that Cb and Cr have the same reference region.
2.2.1 Cache line geometry for minimizing fetch of unused pixels

Since the MAU size is 32 bytes, each access fetches 32 pixels, some of which may not belong to the current reference region as seen in figure 2-1. Our design maps an entire MAU to a single cache line. We repeat the simulations mentioned in section 2.1 for different cache line geometries and tabulate the average fetch overheads as a function of the cache line geometry in table 2.2.

<table>
<thead>
<tr>
<th>Cache line Geometry</th>
<th>Total Overhead</th>
<th>DRAM fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>8×4</td>
<td>4.54</td>
<td>45%</td>
</tr>
<tr>
<td>16×2</td>
<td>5.52</td>
<td>54%</td>
</tr>
<tr>
<td>32×1</td>
<td>7.96</td>
<td>68%</td>
</tr>
</tbody>
</table>

Table 2.2: Average fetch overhead as function of cache line geometry

The 8×4 geometry minimizes the average overhead since it does not fetch too many unnecessary pixels at the edges of the reference regions. When compared with a 32×1 geometry this reduces the amount of unused pixels fetched for a given PU by 60% on average.

Since the fetched MAU are cached, unused pixels may be reused if they fall in the reference region of a neighboring PU. Reference MAUs used for prediction at the right edge of a CTU can be reused when processing CTU to its right. However, the lower CTU gets processed after an entire CTU row in the picture. Due to limited size of the cache, MAUs fetched at the bottom edge will be ejected and are not reused when predicting the lower CTU. When compared to 4×8 MAUs, 8×4 MAUs fetch more reusable pixels on the sides and less unused pixels on the bottom. As seen
2.2. MEMORY ADDRESS MAPPING

<table>
<thead>
<tr>
<th>Encoding Mode</th>
<th>CTU Size</th>
<th>LD</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT BW</td>
<td>Direct 2D</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>MBytes/s</td>
<td></td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Twisted 2D</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>272</td>
<td>690</td>
</tr>
<tr>
<td></td>
<td></td>
<td>227</td>
<td>679</td>
</tr>
<tr>
<td></td>
<td></td>
<td>232</td>
<td>648</td>
</tr>
<tr>
<td></td>
<td></td>
<td>219</td>
<td>667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>183</td>
<td>659</td>
</tr>
<tr>
<td></td>
<td></td>
<td>204</td>
<td>636</td>
</tr>
<tr>
<td>Gain</td>
<td>20%</td>
<td>20%</td>
<td>12%</td>
</tr>
<tr>
<td></td>
<td>3%</td>
<td>3%</td>
<td>2%</td>
</tr>
</tbody>
</table>

Table 2.3: Comparison of Twisted 2D Mapping and Direct 2D Mapping

In figure 2-4(a), this leads to a higher hit-rate. This effect is more pronounced for smaller CTU sizes where hit-rate may increase by up to 12%.

2.2.2 Twisted mapping for minimizing row precharge and activation

The Twisted 2D mapping of figure 2-2 ensures that pixels in different DRAM rows in the same bank are at least 64 pixels away in both vertical and horizontal directions. It is unlikely that inter-prediction of two adjacent pixels will refer to two entries so far apart. Additionally, a single dispatch request issued for motion compensation can at most cover 4 banks. It is possible to keep the corresponding rows in the four banks open and then fetch the required data. These two factors help minimize the number of row changes. Experiments show that twisting leads to a 20% saving in bandwidth over a direct mapping as seen in table 2.3.
2.2.3 Minimizing conflict misses

We use 7 bit line indices when mapping the MAU address to cache lines. Thus a single way in the cache can store 128 MAUs. The mapping between a cache way and a $64 \times 64$ pixel block (128 MAUs) is shown in figure 2-2. We can see that the closest conflicting addresses in a given picture are 64 pixels apart in both horizontal and vertical directions. However, there is a conflict between the same pixel locations across different pictures. Similarly, luma and chroma pixels may conflict. Both these conflicts are tackled by ensuring sufficient associativity in the cache.

Alternative techniques to tackle conflict misses include having separate luma and chroma caches. Similarly offsetting the memory map such that the same x-y in successive frames maps to different cache lines can also reduce conflicts. For our chosen configuration, the added complexity for these techniques outweighed the observed hit-rate improvements.

2.3 Motion Compensation Cache Design

This section describes a high throughput four parallel MC cache architecture. Two main techniques are used to ensure cache throughput: datapath parallelism and hiding the variable DRAM latency with outstanding request queues. Figure 2-3, shows the cache architecture with each of the four parallel paths capable of outputting up to 32 pixels (1 MAU) per cycle.
Figure 2-3: Proposed four-parallel MC cache architecture with 4 independent data-paths. The hazard detection circuit is shown in detail.
2.3.1 Four-Parallel Data Flow

The parallelism in the cache datapath allows up to 4 MAUs in a row to be processed simultaneously. The motion compensation unit connected to the cache processes $16 \times 16$ regions at a time. The MC cache must fetch at most $23 \times 23$ reference region corresponding to this $16 \times 16$ region. This may require up to 7 cycles as shown in figure 2-1. The address translation unit in figure 2-3 reorders the MAUs based on the lowest 2 bits of the column address. This maps each request to a unique datapath and allows us to split the tag register file and cache SRAM into 4 smaller pieces. Note that this design cannot output 2 MAUs in the same column on the same cycle. Thus, our design trades unused flexibility in addressing for smaller tag-register and SRAM sizes.

The cache tags for the missed cache lines are immediately updated when the lines are requested from DRAM. This pre-emptive update ensures that future reads to the same cache line do not result in multiple requests to the DRAM. Note that this behavior is similar to a simple non-blocking cache and does not involve any speculation. Additionally, since the MC cache is a read only cache, there is no need for writeback in case of eviction from the cache.

2.3.2 Queue Management and Hazard Control

Each datapath has independent read and write queues, which help absorb the variable DRAM latency. The 32 deep read queue stores pending requests to the SRAM. The 8 deep write queue stores pending cache misses, which are yet to be resolved by the DRAM. The write queue is shorter because fewer cache misses are expected. Thus,
2.3. MOTION COMPENSATION CACHE DESIGN

Figure 2-4: Cache hit rate as a function of CTU size, cache line geometry, cache-size and associativity. Experiments averaged over six sequences - Basketball Drive, Park Scene, Tennis, Crowd Run, Old Town Cross and Park Joy. The first are Full HD (240 pictures each) and the last three are 4K Ultra HD (120 pictures each). CTU size of 64 is used for the cache-size and associativity experiments.

the cache allows for up to 32 pending requests to the DRAM. At the system level, the latency of fetching the data from the DRAM is hidden by allowing for a separate MV dispatch stage in the pipeline prior to the Prediction stage. Thus, while the reference data of a given block is being fetched, the previous block is undergoing prediction.

Since the cache system allows multiple pending reads, a read queue may have reads for two MAUs that map to the same cache line. If the second MAU results in a cache miss, it may evict the entry of the first MAU before it has been read out by the first read. The hazard control unit in figure 2-3 avoids this by writing the data only after the first read is complete. This is accomplished by checking if the address of the first pending cache miss, matches any address stored in the read queue. Only those entries in the read queue that occur before the entry corresponding to this cache miss are checked.
2.4 Results

2.4.1 Cache Specification

Figure 2-4(b) and figure 2-4(c) shows the hit-rates observed as a function of the cache size and associativity respectively. A cache size of 16K bytes was chosen since it offered a good compromise between size and cache hit-rate. The performance of FIFO replacement is as good as Least Recently Used replacement due to the relatively regular pattern of reference pixel data access. FIFO was chosen because of its simple implementation. The cache associativity of 4 is sufficient to accommodate both RA sequences and the three component planes (Y, Cb, Cr). The specification of the final cache is given in table 2.4.

2.4.2 DRAM Bandwidth and Power Savings

The rate at which data can be accessed from the DRAM depends on 2 factors: the number of bits that the DRAM interface can (theoretically) transfer per unit time and the precharge latency caused by the interaction between requests. The

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache line</td>
<td>32 Bytes (8×4 MAU)</td>
</tr>
<tr>
<td>Cache SRAM</td>
<td>16KB (512 cache lines)</td>
</tr>
<tr>
<td>Set-Associativity</td>
<td>4 Way</td>
</tr>
<tr>
<td>Tag Register File</td>
<td>128×70-bit</td>
</tr>
<tr>
<td>Y/UV Scheme</td>
<td>Unified cache</td>
</tr>
<tr>
<td>Replacement Rule</td>
<td>FIFO Policy</td>
</tr>
<tr>
<td>DRAM Mapping</td>
<td>Twisted 2D Mapping</td>
</tr>
</tbody>
</table>

Table 2.4: Overview of MC Cache Specifications
2.4. RESULTS

Figure 2-5: Comparison of DDR3 bandwidth and power consumption across 3 scenarios. RS mapping maps all the MAUs in a raster scan order. ACT corresponds to the power and bandwidth induced by DRAM Precharge/Activate operations.

precharge latency can be normalized to bandwidth by multiplying with the bitwidth. This normalized figure (called ACT BW) is the bandwidth lost in the precharge and activate cycles - the amount of data that could have been transferred in the cycles that the DRAM was executing row change operation. The other figure, DATA BW refers to the amount of data that needs to be transferred from the DRAM to the decoder per unit time for real-time operation. A better hit-rate reduces the DATA BW and a better memory map reduces the ACT BW. The advantage of defining DATA BW and ACT BW as mentioned above is that \( (\text{DATA BW} + \text{ACT BW}) \) is the minimum bandwidth required at the memory interface to support real-time operation.

The performance of the above-described cache is compared with two reference scenarios: a raster-scan address mapping and no cache and a 16KB cache with the same raster scan address mapping. As seen in figure 2-5(a), using a 16KB cache reduces the Data BW by 55%. The Twisted 2D mapping reduces ACT BW by 71%.
of the ACT BW. Thus, the cache results in a 67% reduction of the total DRAM bandwidth. Using a simplified power consumption model [11] based on the number of accesses, this cache is found to save up to 112mW. This is shown in figure 2-5(b).

Figure 2-5(c) compares the DRAM bandwidth across various encoder settings. Smaller CTU sizes result in a larger bandwidth because of lower hit-rates. Thus, larger CTU sizes such 64×64 can provide smaller external bandwidth at the cost of higher on-chip complexity. In addition, Random Access mode typically has lower hit rate when compared to Low Delay. This behavior is expected because the reference pictures are switched more frequently in the former.

Thus, we have designed an 16K Byte MC cache that results in 67% reduction in DRAM bandwidth and 41% reduction in external DRAM power. We have integrated this cache in a 4k UltraHD video decoder [5]. The following section details a real-time demonstration system for the same.

2.5 4k UltraHD Demo Setup

Figure 2-6 shows a schematic representation of the complete demo system. The HEVC decoder is targeted as peripheral to the main processor in a low power mobile system on chip (SoC). We use two Xilinx FPGA boards (ML605 and KC705) to simulate this SoC. A Microblaze soft-processor on the ML605 acts as the main processor and is connected to various peripherals via the peripheral logic bus (PLB). Multiple compressed bitstreams can be stored on a Compact Flash (CF) connected to the PLB. The software on the Microblaze processor allows us to select the desired sequence at run-time. The drivers provided by Xilinx for the CF card are not fast
2.5. 4K ULTRAHD DEMO SETUP

enough to allow for real-time operation of the entire system. We get around this issue by copying the desired bitstream to the higher speed DRAM before looping the sequence. The HEVC decoder interface is mapped to the processor’s memory and this allows for flexible control of the decoder through software. The DRAM memory interface is provided by the Xilinx Memory Interface Generator (MIG) and we add a custom arbiter around it to arbitrate between the processor and decoder DRAM requests.

Figure 2-6: Block diagram of the 4k UltraHD real-time HEVC decoding demo

The ML605 board does not provide enough IO pins to both interface with our chip and drive four 1080p monitors. Hence we implement the display driver on a separate KC705 board. We have ported the [12] application note from the Spartan-6 to the Kintex-7 FPGA on the KC705 and we use four instances of the same to drive our displays. The decoder sends over the reconstructed pixels from the ML605 to the KC705 in real-time as they are decoded. The 4k resolution at 30 frames per second constitutes a 2.9Gbps bandwidth requirement. We implement a 3.125Gbps high speed serial link between the two boards to support this data transfer. We
use the Aurora 8b/10b [13] link layer due to its relative simplicity and its built-in control flow. The Aurora protocol is capable of clock compensation and can tolerate ±100ppm difference in the reference clock frequencies used on the two boards. In practice we observe that the oscillators on the two boards have a larger mismatch and this leads to a high bit error rate on the link. We get around this issue by generating a reference clock on the ML605 and using it to clock the serial transceivers on both the boards.

The four monitors only accept 1080p sequences at 60 frames per second and thus we cannot display the 4k sequences without some kind of buffering. In addition the decoder outputs the pixels according to memory map specified in section 2.2, one MAU at a time. However the four displays we use, need the pixels in their respective raster-scan orders. We resolve this by buffering the entire frame that is currently being sent by the decoder and displaying the previous frame in the interim period. Thus in all we need to provide space for at least two 4k resolution frames which comes to a total of 24M bytes. This is larger than the available BRAM on the KC705 and we rely on DRAM to implement this frame buffer. The total bandwidth we need to support is 2.9Gbps for the writing the incoming frame and 5.8Gbps for reading the currently displayed frame at 60 frames per second. The total bandwidth of 8.7 Gbps is relatively close to the theoretical DRAM limit of 12.8Gbps. We use memory mapping techniques similar to those shown in section 2.2 to avoid DRAM activate and pre-charge latencies and thus are able to meet the required spec.

The complete decoder chip is implemented in the TSMC 40nm GP process and has a total area of 715K gates and uses 124K bit on-chip SRAM. It achieves 4k UltraHD decoding at 30 frames per second when operating at 200 MHz and consumes
2.5. **4K ULTRAHD DEMO SETUP**

78mW of power for the same. The various decoder modules including the MC cache are shown on the die micrograph in figure 2-7(a). Figure 2-7(b) shows the chip decoding the CrowdRun sequence and displaying it on four 1080p monitors \(^1\).

![Die Micrograph](image1)

![Demo System](image2)

**Figure 2-7:** The die micrograph showing the MC Cache and the complete demo systems showing the decoder chip attached to the ML605 board

\(^1\)A video of the system at the ISSCC 2013 Academic and Industrial Session: [http://player.vimeo.com/video/70360822](http://player.vimeo.com/video/70360822)
Chapter 3

Hardware Friendly Fast Intra Estimation

Embedded video solutions like security cameras are cost-constrained. In such systems, the designer may desire the better quality offered by high-resolution video but may not want to provide the large frame buffer required for inter coding. For example, a 1080p security solution conforming to HEVC level 5 requires a decoded picture of at least 6 frames [14]. This comes to an 18M byte decoded picture buffer. Conventional codecs use an off-chip memory to implement this buffer, which adds to the overall system cost. On the other hand, 18M byte is too large to be implemented in on-chip SRAM. One possible alternative in cost constrained systems is to use intra coding. Since intra coding only uses previously decoded pixels from within the same frame, it is possible to implement an intra coding based encoder with smaller CTU sized buffers, which can be implemented as on-chip memories. In this chapter, we discuss the algorithms and architectures that such an encoder might use for
estimating the intra prediction mode.

3.1 Current Approaches

The intra estimation algorithm employed by the HEVC reference software implementation (HM) and some improvements from recent literature are summarized below.

3.1.1 HM Reference Implementation

The HM intra estimation algorithm [15] consists of two major parts, a depth first search (DFS) to traverse the quad-tree to find the optimal splits and a mode decision algorithm to compare the choices at any given node in the tree.

The tree traversal is implemented by invoking the mode decision algorithm at every node in the quad-tree. This computes the best mode for that particular node assuming that CU is not split and returns the cost of using this mode. This is then compared with the sum of the costs of encoding the best decisions for the four children of that node. The lower amongst these two decides whether the current CU will be split or not. Since the 8×8 CU is the leaf of the CU tree it must additionally decide whether the luma prediction will be split into four 4×4 PU or the kept as a single 8×8 PU. The chroma blocks of an 8×8 CU are never split.

The mode decision algorithm computes cost of a given CU by summing the cost of its luma and chroma components. The algorithm for computing the mode and cost for the luma component is shown in 1. It consists of a rough mode decision (RMD) over all 35 candidates and rate-distortion optimization (RDO) over some shortlisted candidates.
3.1. CURRENT APPROACHES

Algorithm 1 Mode Decision in the HM reference implementation

1: function HM\(_{MD}\)(size, orig\(_{pel}\), neighbors)  \(\triangleright\) HM mode decision
2: \(\text{cost}_{RMD} \leftarrow \text{RMD}(\text{orig\(_{pel}\), [0, ..., 34], neighbors})\)
3: \(\text{modes} \leftarrow \text{SHORTLIST}(\text{cost}_{RMD}, \text{size})\)  \(\triangleright\) 3/8 modes based on PU size
4: \(\text{cost}_{RDO} \leftarrow \text{RDO}(\text{orig\(_{pel}\), modes, neighbors})\)
5: return \((\text{arg min}(\text{cost}_{RDO}), \text{min}(\text{cost}_{RDO}))\)
6: end function

- Rough Mode Decision:

Rough mode decision is performed over all 35 candidates. This pass uses the sum of absolute transformed differences (SATD) of the prediction error along with the bits used to signal the intra mode as the cost metric for shortlisting promising modes. RMD selects 8 modes for the 4×4 and 8×8 PUs and 3 modes for the larger PUs. An additional 3 most probable modes (MPM) may be added if not already included.

Algorithm 2 Rough Mode Decision

1: function RMD(\text{orig\(_{pel}\), modes, neighbors})  \(\triangleright\) SATD based costs for all modes
2: for mode \(\in\) modes do
3: \(\text{pred\(_{pel}\)} \leftarrow \text{PREDICTION}(\text{neighbors})\)
4: \(\text{cost}[\text{mode}] \leftarrow \text{SATD}(\text{orig\(_{pel}\) } - \text{pred\(_{pel}\)}) + \text{bits\(_{mode}\)}\)
5: end for
6: return cost
7: end function

- Rate Distortion Optimization:

The full encoding process consisting of prediction, transform, quantization, inverse quantization, inverse transform, reconstruction and entropy coding is applied to the modes shortlisted by RMD. The reconstructed pixels are compared with the original pixels to find the sum of the squared errors (SSE). This
along with the number of entropy-coded bits is used as the cost metric.

Algorithm 3 Rate Distortion Optimization

1: function RDO(origpel, modes, neighbors) ▷ SSE and entropy based costs
2:   for mode ∈ modes do
3:       predpel ← PREDICTION(neighbors)
4:       coef ← QUANT('TRANSFORM(origpel − predpel))
5:       reconpel ← predpel + TRANSFORM−1(DEQUANT(coef))
6:       bits ← |ENTROPY(coef)|
7:       dist ← Σ (origpel − reconpel)²
8:       cost[mode] ← dist + λ × bits
9:   end for
10: end function

For the chroma components, RMD is skipped and only the RDO step is performed because there are at most five possible candidates as explained in section 1.2.1.

In both the RMD and RDO steps, reconstructed pixels from the previously encoded PUs are used as neighbors for future PUs. This serial dependency makes pipelining the intra estimation algorithm hard [2]. Previous work for AVC/H.264 encoders either used original pixels as proxy for these reconstructed pixels [16] or stalled the encoder [17] to allow for this dependency. This is not a feasible option for an HEVC encoder because the larger PU sizes like 64×64 and 32×32 take can cause up to 16× longer stalls in the encoder pipeline. On the other hand, our experiments show that using original pixels results in up to 3% coding loss for the intra coded frames.
3.1. CURRENT APPROACHES

3.1.2 Related Work

Table 3.1 summarizes the current state of the art in development of intra estimation algorithms.

<table>
<thead>
<tr>
<th>Category</th>
<th>Ref.</th>
<th>Contribution</th>
<th>Open Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>[18]</td>
<td>35 mode RMD and simplified RDO based on grouping modes based on RMD costs. (1% BD-rate loss)</td>
<td>RMD as complex as HM Multiple candidate RDO</td>
</tr>
<tr>
<td>Decision</td>
<td></td>
<td>Gradient based pre-processing. Up to 12 mode RMD and 5 mode RDO. (0.74% BD-rate loss)</td>
<td>Multiple candidate RDO</td>
</tr>
<tr>
<td>DFS</td>
<td>[19]</td>
<td>Bayesian statistical model. Split CU after RMD or prune children after RDO. (2% BD-rate loss)</td>
<td>Needs online training phase to train the model</td>
</tr>
<tr>
<td>Pruning</td>
<td></td>
<td>Cost information of neighboring CUs used to implement early termination. (1.74% BD-rate loss)</td>
<td>Frame content dependent thresholds</td>
</tr>
<tr>
<td>Hardware</td>
<td>[2]</td>
<td>ASIC implementation for 8k UltraHD encoder. 12 mode RMD and 1 mode RDO. (8% BD-rate loss [Low Delay])</td>
<td>Need to improve coding efficiency. No reconstructed pixel feedback</td>
</tr>
<tr>
<td></td>
<td>[21]</td>
<td>FPGA implementation of all intra encoder. Mode decision based on [1]. (~17% BD-rate loss)</td>
<td>Need to improve coding efficiency</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of published literature on intra estimation algorithms

Published work on improving the mode decision algorithm for intra estimation also follows the same two-pass mode decision used in the HM algorithm. [18] reduces the number of RDO candidates based on the number of minima in the RMD costs of all the 35 modes. [1] implements a gradient based approach. They first calculate a gradient based mode histogram for the PU under consideration. Based on this they select fewer modes for the RMD and RDO operations. Our approach is similar to [1]
and we detail the similarities and differences in 3.2. We also implement their mode decision algorithm as a control to benchmark our own.

Additionally there has been some work on pruning the DFS to reduce the search space. [19] uses a Bayesian model to prune the tree while [20] uses the decisions of neighboring PUs. While we do not propose any explicit algorithmic schemes for tree pruning we do ensure that our proposed architecture is consistent with the DFS ordering such that early termination and pruning can be added in later by programming the appropriate threshold values.

While most existing algorithms are meant for software implementations, [2] demonstrates a real-time 8K UltraHD encoder with a 12 candidate fast intra prediction and without reconstructed pixel feedback. They achieve an 8% BD-rate loss from the RMD and RDO optimizations for intra [22]. Note, however, that the above number is presented for a low delay (LD) configuration, which also uses inter frames. In order to draw a fair comparison with their work, we have implemented their algorithm and run it under an All Intra (AI) configuration. [21] implements an AI HEVC encoder on an Altera FPGA. They use a simplified version of the algorithm presented in [1] and achieve approximately 17% coding loss.

3.2 Algorithm Design

We now detail our hardware friendly intra estimation algorithm. The goal of our design is to reduce the complexity of the algorithm while maintaining the coding efficiency. Our design follows the general theme of [1]. First a gradient based mode pruning (GBMP) is performed to select a set of likely modes. These selected modes
are then passed through a fast mode decision step (with simplified RMD and RDO) to select the final mode.

### 3.2.1 Gradient Based Mode Pruning

Angular intra prediction computes the value of the predicted pixels by copying the neighboring pixels along the direction indicated by the intra prediction mode. As a result of this, the gradient of the predicted pixels is maximum along the direction perpendicular to the intra mode. If the predicted pixels are a fair representation of the original pixels, then the same trend will also be true for the original pixels. We use this motivating observation to prune unlikely intra modes before the RMD step. The GBMP algorithm is shown in figure 3-1.

For all the pixels in a given PU we first find the per-pixel horizontal ($g_x$) and ($g_y$) vertical gradients by applying the corresponding Sobel masks via an element-wise dot product. In order to simplify the hardware implementation, we only use pixels within a $32 \times 32$ PU to perform this computation. We avoid fetching extra pixels across the $32 \times 32$ PU boundary by padding the gradient values for the border pixels from their neighbors.

We then compute the per-pixel weights ($|g_x| + |g_y|$) and intra mode bins. Since the intra mode is perpendicular to the direction of the maximum gradient, [1] computes the per pixel intra mode bin by computing $90^\circ + \tan^{-1}\left(\frac{g_y}{g_x}\right)$. Implementing this computation in hardware, as is, would be challenging due to the $\tan^{-1}(\cdot)$ and division operations. In reality, we do not need to perform the above operation exactly. Every intra mode corresponds to a range of gradient angles and we only need to compute the correct range that the current gradient lies in.
We first note that, 

\[ \text{mode}_z = 90^\circ + \tan^{-1}\left( \frac{g_y}{g_x} \right) \]

\[ \therefore \tan(\text{mode}_z) = -\frac{g_x}{g_y} \]

Knowing the sign and the relative magnitudes of \( g_x \) and \( g_y \) it is easy to find the 45\(^\circ\) range the correct mode lies in. The values of \(|\tan(\cdot)|\) for all the intra angles are
\( \frac{n}{32} \) for \( n \in \{0, 2, 5, 9, 13, 17, 21, 26, 32\} \) [14]. Thus, we can find the correct mode by comparing \( |g_x| \times 32 \) with \( |g_y| \times n \) for each of the aforementioned \( n \). This method only needs constant multipliers, which can then be further amortized using multiple constant multiplication (MCM) as shown in section 3.3.5. Note that since each of these steps was just an arithmetic simplification, there is no loss in accuracy associated with our method.

Once the per pixel weights and mode bins are computed, we compute a histogram of the modes with each pixel contributing its weight to its corresponding mode bin. We then select the two best values from the histogram and use them as the angular direction candidates.

We note two interesting properties of this histogram computation from a hardware reuse perspective.

1. The histogram computation can be composed across PU sizes. The histogram of an 8×8 is just the element-wise sum of the histograms of its four constituent 4×4 blocks. Thus the histogram computation need not be repeated at the various block sizes and we can reuse our previous computations.

2. The histogram that we compute is very similar to the histogram of oriented gradients (HOGS) feature used in machine learning. [23] A system that both encodes a video stream and performs some sort of signal processing (e.g. face detection) could potentially reuse this computation for both purposes.

Gradient based mode pruning is only performed on the luma pixels in the image. This is because the chroma angular mode is fixed to be either 10, 26, 34 or luma mode and hence no separate mode pruning is necessary.
3.2.2 Fast Mode Decision

Both [1] and [2] use up to 12 modes for RMD. In order to reduce the complexity of the RMD process we restrict ourselves to just 6 modes and obtain nearly similar or better performance. In addition to the 2 modes selected previously we also select DC, Planar, vertical and horizontal modes by default. Figure 3-2 shows the distribution of intra modes selected by the HM algorithm. The relatively high likelihood of these modes being picked justifies their inclusion by default.

Figure 3-2: Distribution of intra modes estimated by HM. Modes included by default in fast mode decision are shown in red
3.2. **ALGORITHM DESIGN**

The RDO operation in intra estimation is much more expensive than the RMD. This is because while the RMD utilizes only a Hadamard transform for cost estimation, the full RDO process also includes a forward and inverse DCT, CABAC bit estimation and an SSE operation for computing the distortion. Thus the multiple iterations of RDO pose a significant challenge to real-time operation. It would be ideal to find an intra estimation algorithm that only consists of an RMD pass.

Unfortunately, while the RMD cost is a reasonable indicator of the optimal mode for a given PU size it performs very poorly when comparing across CU sizes. RMD can effectively indicate which mode to use at a given node in the quad-tree but it is a very poor discriminator of whether a CU must be split or not. Experiments show that using the RMD cost in place of the RDO cost results in over 20% BD-rate loss. Such a high loss would defeat the purpose of using HEVC, since on average HEVC offers 22% coding gain over AVC/H.264 for all-intra (AI) sequences.

We balance these conflicting requirements by using the RMD to find the optimum mode decision among the 35 candidates and then performing a single RDO iteration on the best mode to enable CU split decision as shown in 4.

---

**Algorithm 4 Fast Mode Decision**

```plaintext
1: function HARDWAREMD(size, origpel, neighbors)
2:   modesang ← MODEPRUNING(origpel)  \>
> Gradient based mode pruning
3:   modes ← modesang + [DC, PLANAR, HOR, VER]
4:   cost ← RMD(origpel, modes, neighbors)
5:   mode ← arg min(cost)
6:   cost ← RDO(origpel, [mode], neighbors)
7:   return (mode, cost[mode])
8: end function
```

We have modified the distortion computation in the RDO process to use the SSE
of the transform coefficients before and after quantization. This is nearly the same as the original distortion metric since most of the loss is introduced by the quantization process itself. As a result of this change, we only need to perform reconstruction for the pixels that act as references for neighboring PU and we can avoid performing it on the other pixels in the modes that may not be selected finally. Finally, since the 64×64 PU mode is rarely chosen we have turned it off to reduce complexity.

3.2.3 Results

Since the target application for the estimation algorithm is a real-time HEVC encoder, we have changed the encoder configuration to reflect the same. All results reported in this section are for the Al configuration. To simplify the encoder implementation we have disabled TU splitting, Transform Skip and RDOQ tools. Additionally we use tiles as an additional mode for parallelism and allow for 4 vertical tiles in 1080p sequences. Since all of these constraints are imposed by factors outside the scope of the intra estimation algorithm design, we use these settings for both the reference anchor and our implementation. We test all algorithms across four 1080p sequences (Cactus, Kimono, ParkScene, Tennis) and two 720p sequences (Johnny, KristenAndSara). We use 100 frames for the larger sequences and 200 frames for the smaller ones. In addition we implement the algorithms mentioned in [1] and [2] to compare our algorithm.

Figure 3-3 shows that on average we see about 6% increase in BD-rate from our algorithm on the 1080p sequences. We also tested other variants of our algorithm to understand the cause of this coding loss. The detailed results for these test
3.2. **Algorithm Design**

The open-loop variant of our algorithm uses the exact same mode decision algorithm but uses original pixels instead of the reconstructed pixels resulting in a maximum of 2.9% coding loss. The fast-intra (8) variant uses 4 candidates from the GBMP instead of 2 but only provides a modest 0.3% coding loss.
gain on average. The RDO variant runs an RDO pass over all 6 RMD candidates. This gives an average 4.3% coding gain on the 1080p sequences. This shows that most of the coding loss is caused by the lack of the RDO and that we have reached a point of diminishing returns on improving the RMD algorithm.

Finally, we compare our work to [1] and [2]. Since [2] also targets hardware encoders, they too run a simplified RDO with just one mode. In order to draw a fair comparison between all three, we replace the CFBAC bit estimator in [2] with HM’s CABAC based bit-estimator (improving their performance). [1] originally targets software implementations and we compare our work with them through two simulations. In the first case we run both algorithms with a single mode RDO and in the second case we run both algorithms with the number of RDO modes specified by their algorithm. While running these tests our algorithm pads the gradient at the boundaries, 64×64 PU is disabled and quantization based distortion is used as explained in the previous section. The relevant settings are not modified for the control algorithms. Table 3.3 shows that our algorithm gives comparable or better performance with a 50% reduction in the complexity of RMD.

### 3.3 Architecture and Implementation

An architecture for implementing the algorithm described in the previous section is presented below.
## Table 3.2: Comparison of BD rate across various QPs and sequences

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Algorithm</th>
<th>QP22</th>
<th>QP27</th>
<th>QP32</th>
<th>QP37</th>
<th>BD rate</th>
<th>Δ rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PSNR</td>
<td>Rate</td>
<td>PSNR</td>
<td>Rate</td>
<td>PSNR</td>
<td>Rate</td>
</tr>
<tr>
<td>Cactus</td>
<td>Anchor</td>
<td>40.57</td>
<td>109454.32</td>
<td>37.86</td>
<td>51058.85</td>
<td>35.58</td>
<td>27814.25</td>
</tr>
<tr>
<td></td>
<td>Open Loop</td>
<td>40.38</td>
<td>109592.41</td>
<td>37.73</td>
<td>52514.22</td>
<td>35.43</td>
<td>29162.04</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (6)</td>
<td>40.39</td>
<td>109327.29</td>
<td>37.74</td>
<td>52145.76</td>
<td>35.44</td>
<td>26851.52</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (8)</td>
<td>40.39</td>
<td>109146.63</td>
<td>37.74</td>
<td>51930.58</td>
<td>35.44</td>
<td>26688.71</td>
</tr>
<tr>
<td></td>
<td>RDO</td>
<td>40.47</td>
<td>109068.78</td>
<td>37.81</td>
<td>51610.85</td>
<td>35.53</td>
<td>28324.57</td>
</tr>
<tr>
<td>Kimono</td>
<td>Anchor</td>
<td>42.99</td>
<td>17852.07</td>
<td>41.65</td>
<td>10229.49</td>
<td>39.82</td>
<td>6136.87</td>
</tr>
<tr>
<td></td>
<td>Open Loop</td>
<td>42.91</td>
<td>18267.75</td>
<td>41.55</td>
<td>10552.9</td>
<td>39.69</td>
<td>6406.72</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (6)</td>
<td>42.91</td>
<td>18194.89</td>
<td>41.56</td>
<td>10464.55</td>
<td>39.7</td>
<td>6323.94</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (8)</td>
<td>42.91</td>
<td>18218.42</td>
<td>41.55</td>
<td>10468.08</td>
<td>39.69</td>
<td>6317.55</td>
</tr>
<tr>
<td></td>
<td>RDO</td>
<td>42.98</td>
<td>17855.25</td>
<td>41.64</td>
<td>10274.44</td>
<td>39.82</td>
<td>6182.58</td>
</tr>
<tr>
<td>ParkScene</td>
<td>Anchor</td>
<td>41.63</td>
<td>52760.48</td>
<td>38.59</td>
<td>28660.85</td>
<td>35.62</td>
<td>14952.04</td>
</tr>
<tr>
<td></td>
<td>Open Loop</td>
<td>41.43</td>
<td>52937.41</td>
<td>38.41</td>
<td>28924.28</td>
<td>35.45</td>
<td>15210.18</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (6)</td>
<td>41.44</td>
<td>52801.4</td>
<td>38.42</td>
<td>28808.42</td>
<td>35.47</td>
<td>15132.07</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (8)</td>
<td>41.44</td>
<td>52827.2</td>
<td>38.42</td>
<td>28821.28</td>
<td>35.47</td>
<td>15136.59</td>
</tr>
<tr>
<td></td>
<td>RDO</td>
<td>41.54</td>
<td>52487.61</td>
<td>38.53</td>
<td>28582.02</td>
<td>35.58</td>
<td>14943.49</td>
</tr>
<tr>
<td>Tennis</td>
<td>Anchor</td>
<td>42.08</td>
<td>22246.49</td>
<td>40.31</td>
<td>11722.55</td>
<td>38.27</td>
<td>6188.54</td>
</tr>
<tr>
<td></td>
<td>Open Loop</td>
<td>41.99</td>
<td>22660.52</td>
<td>40.22</td>
<td>12287.73</td>
<td>38.15</td>
<td>6463.73</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (6)</td>
<td>41.99</td>
<td>22646.93</td>
<td>40.22</td>
<td>12071.3</td>
<td>38.15</td>
<td>6480.3</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (8)</td>
<td>41.99</td>
<td>22614.94</td>
<td>40.22</td>
<td>12048.5</td>
<td>38.16</td>
<td>6468.58</td>
</tr>
<tr>
<td></td>
<td>RDO</td>
<td>42.04</td>
<td>22294.88</td>
<td>40.28</td>
<td>11829.61</td>
<td>38.24</td>
<td>6271.37</td>
</tr>
<tr>
<td>Johnny</td>
<td>Anchor</td>
<td>44.07</td>
<td>19879.85</td>
<td>41.9</td>
<td>11123.4</td>
<td>39.57</td>
<td>6409.95</td>
</tr>
<tr>
<td></td>
<td>Open Loop</td>
<td>43.94</td>
<td>20642.16</td>
<td>41.75</td>
<td>11834.36</td>
<td>39.37</td>
<td>7002.58</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (6)</td>
<td>43.94</td>
<td>20419.92</td>
<td>41.76</td>
<td>11620.65</td>
<td>39.4</td>
<td>6826.26</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (8)</td>
<td>43.94</td>
<td>20340</td>
<td>41.76</td>
<td>11569</td>
<td>39.41</td>
<td>6790.44</td>
</tr>
<tr>
<td></td>
<td>RDO</td>
<td>44.01</td>
<td>20115.66</td>
<td>41.84</td>
<td>11337.55</td>
<td>39.51</td>
<td>6556.12</td>
</tr>
<tr>
<td>KristenSara</td>
<td>Anchor</td>
<td>44.46</td>
<td>21812.14</td>
<td>42.15</td>
<td>13022.34</td>
<td>39.53</td>
<td>7878.69</td>
</tr>
<tr>
<td></td>
<td>Open Loop</td>
<td>44.29</td>
<td>22634.43</td>
<td>41.97</td>
<td>13771.88</td>
<td>39.29</td>
<td>8507.3</td>
</tr>
<tr>
<td></td>
<td>Fast Intra (6)</td>
<td>44.29</td>
<td>22408.17</td>
<td>41.98</td>
<td>13585.58</td>
<td>39.32</td>
<td>8359.29</td>
</tr>
</tbody>
</table>
### 3.3.1 Challenges and Design Goals

Before describing the implementation in detail we first outline the key design goals and challenges that our implementation must address.

**Reconstructed Pixel Feedback:** Reconstructed pixel feedback gives up to 3% coding gain. However, implementing it in hardware results in a loss of parallelism because the next PU must now wait for the estimation of the previous PU. This added latency can result in pipeline stalls as seen in figure 3-5 and must be accounted for in the design.

**RDO support:** RDO is necessary for reasonable coding performance. The increased complexity of the HEVC transform makes this a challenging proposition.

**Support for Tiling:** Many components of an HEVC encoder such as the CABAC
and Motion Estimation can benefit from the added parallelism enabled by tiles. In order to realize these benefits the intra estimation block must also be designed to support tiles.

**Support for early termination:** Early termination can offer significant power reduction if the correct thresholds are known a priori. Previous work on HEVC encoders, runs multiple PU sizes in parallel. With such a pipeline schedule it is not possible to exploit early termination. We design our intra estimation pipeline to serialize the processing of the various PU sizes similar to work done in [24] and [25] on AVC/H.264, so that unlikely PU sizes can be pruned by the use of programmable thresholds.

The main trade-off in such a serial implementation is the granularity at which parallelism can be exploited. For example if running four PU sizes in parallel at 2 pixels/cycle is sufficient to meet real-time constraints, a single serial pipeline must process all four PU sizes at 8 pixels/cycle to meet the same constraints. This involves cleverly exploiting PU level parallelism to provide an area-efficient design. The large variety of intra PU sizes in HEVC makes the design of such a high-throughput flexible pipeline a challenging problem.

**Throughput:** We support FullHD (1080p) video at 30 frames per second when the encoder is run at 50 MHz. This translates to an average throughput close to 2 pixels/cycle for real-time operation.
3.3.2 Tiles, Multi-threading and Feedback

Tiling is a new mode of parallelism added in HEVC. An example of a video sequence with four vertical tiles is shown in figure 3-4. The four CTUs in these tiles can undergo estimation independent of each other as neighboring pixel dependencies are not allowed across tile boundaries. The common approach to using the parallelism afforded by tiling is to run independent parallel threads in software. In hardware this would roughly translate to running four intra estimation engines in parallel. In this work we propose using the parallelism provided by tiles to mitigate the latency of the reconstructed pixel feedback.

Figure 3-4: An example video sequence with 4 vertical tiles. The four tiles can be specified as any rectangular region. We size them at 512 pixels in width with the last tile being smaller if the frame width is less than 2048 pixels. The 4 CTU shown in red can be processed independently of each other.
Consider a four stage pipeline for performing intra-estimation as shown in figure 3-5. Such a pipeline would stall for 75% of the time due to the neighbor dependency and would need to be designed for 4x throughput. On the other hand if we used the same idle pipeline stages to run PUs in different tiles we could get back 100% hardware utilization and would not need any over-design to meet the required spec.

In reality we are unable to meet perfect utilization due to the variable PU sizes that need to be serialized as seen in the figure 3-5. The pipeline is stalled for two reasons: a smaller PU is waiting for a larger PU to vacate a future stage in the pipeline or a larger PU is not ready yet and hence the pipeline stage remains idle.

Our mode decision architecture involves four stages as explained in section 3.3.6 and hence we configure the encoder to use four tiles for 1080p sequences.

3.3.3 Top Level Pipeline

The top level intra pipeline consists of three stages: original pixel prefetch, gradient based mode pruning (GBMP) and mode decision as shown in figure 3-7. The prefetch stage copies the original pixels from a larger frame level buffer to the local original pixel (OrgPel) SRAMs in preparation for the estimation of the next block. Similarly previously reconstructed pixels are copied to the neighbor memory. The original pixels are then used for GBMP which selects the candidates for intra estimation and stores them in the candidate SRAM. The mode decision block uses these candidates and the original pixels to estimate the intra modes and the split structure for the current block. Finally it updates the neighbor SRAMs with reconstructed pixels that will acts as neighbors for the next block.

The largest intra PU size supported by our algorithm is 32x32 and we use that
Figure 3-5: Effect of Multi-threading on throughput. The top figure shows a single threaded implementation and demonstrates the stall caused by the neighbor dependency. Using 4 tiles we can potentially achieve 100% utilization by processing a different tile in the stall period as seen in the middle figure. The real utilization is lower because the variable PU size prevents perfect pipelining as shown in the bottom figure.
3.3. ARCHITECTURE AND IMPLEMENTATION

Figure 3-6: Top level of the intra estimation block showing the various computation blocks and memories

as the granularity of the top level pipeline. One $32 \times 32$ PU from each of four tiles in active in prefetch and GBMP stages while the previous set of four PUs undergoes mode decision as shown in figure 3-7. We have serialized the first 2 stages of the design such that the GBMP stage can operate on the luma pixels with a throughput of 2 pixels/cycle and the prefetch stage fetches both the luma and chroma pixels in the remaining cycles. As a result of this the prefetch stage must operate at a throughput of 6 pixels/cycle. Our frame-level original pixel store offers a read throughput of 16 pixels/cycle and the OrgPel SRAMs are designed to offer a minimum write throughput of 8 pixel/cycle. Thus we can comfortably complete both the prefetch and GBMP operations in the designated cycle budget. The cycle budget for the mode decision is explained in section 3.3.6.
3.3.4 Prefetch and Original Pixel SRAM

The prefetch operation fetches original pixels that are used by both GBMP and mode decision. Both these processes have very different read throughputs and access patterns. Mode decision happens on a different set of four 32×32 PU and hence a ping-pong buffer is needed to store the current fetched pixels while allowing mode decision to parallel work on the last set of fetched PUs. Thus this ping-ponged memory must be sized to store a total of eight 32×32 PUs which comes to a total of 12K bytes of storage. GBMP on the other hand only needs to work on the luma color plane one 32×32 PU which comes to a mere 1K byte in comparison.

It is theoretically possible to source the GBMP pixels from the ping-pong buffer used to store the original pixels however adding a secondary storage buffer for the GBMP leads to a much simpler design. This is because GBMP requires a 6 pixel/cycle readout at offset locations as shown in figure 3-8 whereas mode decision always needs 16 pixel/cycle aligned accesses. The only way to allow both types of accesses from same memory is to use multiple SRAM banks and to allow GBMP to fetch excess pixels (similar to the MAU quantization seen in section 2.1). This
results in an area and power inefficient solution and instead we allocate a separate smaller one port register files to GBMP.

Figure 3-8: An example of a misaligned 6 pixel gradient based mode pruning access

Mode decision itself needs original pixels prior to SATD in RMD and prior to the transform in RDO. While both accesses have a 16 pixel/cycle throughput they can have very different access patterns based on the PU size as shown in figure 3-9. Accommodating both in a single SRAM array needs 8 banks with a 4 pixel readout each. Instead we use 2 banks with 8 pixel readouts to support the RMD operation and a separate two port register file dedicated to RDO. The RDO register file is filed in when the pixels are read for the RMD and is sized for 2 32×32 PUs to allow for the latency between RMD and RDO.

Table 3.4 compares the normalized area of the three alternatives mentioned above and shows that the smallest area option is one with the 25% more storage due to read/write circuit overheads. Moreover having separate banks dramatically reduces the control complexity and reduces the fetch of unnecessary pixel for GBMP.
Figure 3-9: The multiple read pattern required to support both RMD and RDO operations need 8 SRAM banks with 4 pixel readouts. Using 2 banks for RMD and a separate buffer for RDO offers a better area trade-off between cell area and read/write circuits in the SRAM.

### 3.3.5 Gradient based Mode Pruning Architecture and Circuits

Figure 3-10 shows the top level architecture for the GBMP block. This block computes the 2 most likely directional intra candidates from gradients in the original pixels. The OrgPel SRAM architecture is discussed in section 3.3.4. The gradient compute block uses these pixels to compute the horizontal and vertical gradients which are then converted into mode histograms. The selection block chooses the two modes with the highest weights and updates mode histograms for the larger PU sizes.
3.3. ARCHITECTURE AND IMPLEMENTATION

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SRAM</th>
<th>Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Common</td>
<td>2×8 1-Port SRAM (32b×192 words)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Mode Decision</td>
<td>2×8 1-Port SRAM (32b×192 words)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GBMP</td>
<td>4 1-Port RF (16b×128 words)</td>
<td>0.11</td>
<td>1.11</td>
</tr>
<tr>
<td>RMD</td>
<td>2×2 Single Port DRAM (64b×384 words)</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>RDO</td>
<td>2 2-Port RF (64b×128 words)</td>
<td>0.20</td>
<td>0.95</td>
</tr>
<tr>
<td>GBMP</td>
<td>4 1-Port RF (16b×128 words)</td>
<td>0.11</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.4: Comparison of OrgPel SRAM configurations. Factor of 2× indicates that the bank is ping-ponged.

so that the gradient, mode and weight computations can be reused. The selected modes are finally written into the candidate SRAMs so that they can be used by the mode decision block.

Gradient Based Mode Pruning Architecture

Figure 3-10: Gradient based mode pruning design consisting of histogram and reuse registers, a separate original pixel SRAM and computation blocks

A 4×4 PU region needs to access a 6×6 pixel region in order to perform gradient computation. This leads to 2.25× fetch overhead as the same pixels need to be fetched to from the GBMP OrgPel SRAM multiple times. We use a small set of
registers to cache some of the recently read pixels as shown in figure 3-11. A total of 32 8-bit registers are used to implement this caching (12 horizontal and 20 vertical). Once data is populated in these registers the successive PUs can access it directly without going to the SRAM.

**Original Pixel reuse**

![Original Pixel Reuse](image)

Figure 3-11: Deployment of vertical and horizontal original pixel reuse registers

As seen in figure 3-12 the above register reuse scheme reduces the fetch overhead by 80% with minimal hardware overhead.
Gradient Computation

The horizontal and vertical gradient of $p_{x,y}$ are given by:

$$g_x = (p_{x+1,y-1} + 2 \times p_{x+1,y} + p_{x+1,y+1}) - (p_{x-1,y-1} + p_{x-1,y} + p_{x-1,y+1})$$

$$g_y = (p_{x-1,y+1} + 2 \times p_{x,y+1} + p_{x+1,y+1}) - (p_{x-1,y-1} + p_{x,y-1} + p_{x+1,y-1})$$

In order to minimize the duplicate fetching of pixel data the GBMP block computes the partial sums $(p_{x-1,y} + 2 \times p_{x,y} + p_{x+1,y})$ and $(p_{x+1,y} - p_{x-1,y})$ four pixels at a time and accumulates them over 6 cycles as shown in figure 3-13.

Once these gradient have been computed they are converted to modes and weights.
CHAPTER 3. HARDWARE FRIENDLY FAST INTRA ESTIMATION

Gradient Computation

Figure 3-13: Parallel computation of 4 vertical and horizontal gradients

The mode computation is achieved by a multiple constant multiplication (MCM) block that computes all possible values of $|g_y| \times n$ for $n \in \{2, 5, 9, 13, 17, 21, 26, 32\}$. This is then compared with $|g_y| \times 32$ and the result is passed through a priority encoder to find the bin in which the gradient lies as shown in figure 3-14. The MCM block shares common arithmetic operations to implement the 8 constant multiplication using just 4 adders. The MCM block is generated using the Spiral tool [26].

Best mode selection

Once all the modes are computed and the histogram is tabulated we need to find the two modes with the largest weights. One approach would be to sort the entire histogram, but this is inefficient since we only need the 2 largest values. Instead we use the mode selection circuit shown in figure 3-15. This mode selector is built using 4-to-2 compare circuits that give the best two modes when given four inputs. We
use four such blocks over 2 cycles to compute the best two modes. We have adjusted the parallelism in this computation to meet the overall 2 pixels/cycle throughput on the GBMP block. The current histogram values are updated into the histogram of the next PU size in parallel with the mode selection so that the gradient, mode and weight computation can be reused.

Figure 3-16 illustrates the GBMP pipeline for an 8×8 PU. We see that PU4₁ and PU4₃ blocks need fewer cycles due to the reuse registers. When the last 8×8 PU of a 16×16 or 32×32 PU is undergoing GBMP, additional select and accumulate operations are scheduled to accommodate the larger PUs.
CHAPTER 3. HARDWARE FRIENDLY FAST INTRA ESTIMATION

3.3.6 Mode Decision Architecture

The mode decision block uses the original pixels fetched by the prefetch block and the candidates generated by the GBMP block and generates the best possible split structure and PU modes for four 32x32 PUs. The mode decision unit consists of RMD and RDO blocks as shown in figure 3-17.
The RMD block performs 6 parallel predictions and SATD computations on all the RMD candidates. Based on the PU size, the SATD operation consists of either a 4x4 or an 8x8 2D Hadamard transform. In a 2D transform the column transforms cannot proceed before all the row transforms are completed. This restricts the options available for pipelining the RMD operation. Since a single cycle 8x8 Hadamard transform would result in a 64 pixel/cycle throughput the only reasonable option available is to split the row and column transforms into separate pipeline stages. Thus we split the RMD operation into 2 pipeline stages. The first stage computes the prediction, subtracts the original pixels and performs the row transforms. The second stage performs the column transforms and accumulates the transform values to find the RMD cost.
Based on the RMD cost, the best mode is chosen for RDO. In RDO the prediction is followed by a full forward and inverse transform. Once again the 2D transform must be split into separate pipeline stages. The HEVC forward transform performs the row transforms before the column transforms and the inverse transform performs the column transforms before the row transforms. This allows us to merge the second stage of the forward transform and the first stage of the inverse transform into a common pipeline stage.

Since we have modified the RDO operation to use an SSE based on the quantized values of the coefficients, we only need to perform the full inverse transform on the neighbor pixels. We can simply accumulate partial products for the neighbor pixels as we perform the column inverse transforms and avoid doing a costly inverse row transform.

Thus the RDO operation can also be performed in two pipeline stages the first performs the prediction and row forward transform and the second performs the column forward transform and a partial inverse transform.

Since the mode-decision block must be run four time per pixel (once each for 32×32, 16×16, 8×8, 4×4 PU sizes) the throughput of the mode decision unit needs to be at least 4× the nominal throughput of 2 pixels/cycle. Moreover pipeline stalls can occur as shown in figure 3-5 when different sized PUs follow one another. To account for these stalls we have conservatively doubled the throughput of the mode decision block resulting in a total throughput of 16 pixels/cycle.
Chapter 4

Conclusion

4.1 Thesis Summary

This thesis presents two solutions for addressing complexity in HEVC. We have proposed a motion compensation cache for HEVC decoders and an intra estimation algorithm and architecture for HEVC encoders. We now summarize the key lessons learned in the design of these solutions.

1. Motion Compensation Cache Design

MAU quantization: While the increased size of the interpolation filters is an important cause of the increase in the motion compensation bandwidth, we realized that the quantization of the requests due to DRAM MAUs is also an equally important contributor. An effective cache must address both these issues.

Cacheline geometry (8×4 vs. 4×8): The motion compensation operation
by itself is not biased towards either of the two geometries. However the CTU raster scan order results in the 8×4 geometry performing better. This conclusion is also supported by the fact that the advantage is maximum for the smallest 16×16 CTUs.

**Hazards:** It is very interesting to note that even a read only cache is susceptible to pipeline hazards. Moreover the infrequent nature of these hazards makes it important that the designer account for them before hand since they might be missed in short verification tests.

**Power savings:** The MC cache was designed with the goal of reducing the MC bandwidth in order to allow for real-time decoder operation. We note that such caches are beneficial even when the external memory has sufficient bandwidth since they result in power savings.

2. **Intra Estimation Algorithm Design**

**Need for RDO:** RDO is much more computationally intensive than the RMD operation. However it is necessary since RMD cannot be used for CU split decisions.

**Default Modes:** HEVC is biased towards prediction of DC, planar, horizontal and vertical modes. Incorporating this prior knowledge in our encoder can help reduce encoder complexity.

**Reconstructed Pixel Feedback:** Reconstructed pixel feedback results in significant coding gain. It is an attractive option for encoders that already use tiling for parallelizing other operations like CABAC or Motion Estimation.
3. Intra Estimation Architecture Design

**Register Caching:** Register based caching is extremely effective and even a few registers can have a dramatic impact on the required SRAM throughput. This reduced throughput can translate to both lower power (fewer reads to the SRAM) and lower area (smaller read ports on the SRAM).

**SRAM Port Optimization:** While it is possible to support very flexible read patterns by using more SRAM banks with smaller depths, in reality it might be more area efficient to just duplicate some memory and reduce the read flexibility. This is especially beneficial when the SRAM area is dominated by the read/write circuits.

**Register Read Optimization:** When read patterns to a register bank are known a priori simplified read circuits can be constructed that trade-off flexibility for area.

**Natural data compression:** Many computations are naturally compressive. For example SATD compresses an error matrix into a single number. Pipelining the computation at these points results in smaller pipeline registers.

4.2 Future Work

A few directions for extending this work are given below:

1. A major contributor to the DRAM power was the standby power. A custom memory controller can better use the standby modes available to the DRAM.
Similarly lower leakage memories like mobile LPDDR can be explored to reduce this even further. [27]

2. Embedded memory solutions like eDRAM can be added as a much larger last level cache [28] to improve hit rates even further.

3. Early termination and CU pruning techniques discussed in section 3.1.2 can be explored to avoid unlikely modes and save power.

4. Simpler alternatives for RDO can significantly reduce the encoder complexity. Machine learning techniques like function approximation can be explored to find better approximations to the RDO cost.

5. HEVC intra coding can also be used for compressing still images [29] and performs better than algorithms like JPEG. This work can be adapted to developing a low power coprocessor for compressing still images in addition to video.

6. Techniques such as approximate arithmetic and loop perforation [30] can be explored for further reducing encoder complexity.
Bibliography


